TP d'introduction à Python
The Game of Life

Pierre-Marie de Rodat (LSE)

20 novembre 2012

Table des matières

1 Introduction

Bien connu des informaticiens, le Jeu de la Vie [1] n'a rien d'un jeu vidéo.

Il s'agit d'étudier l'évolution d'un système simple à l'aide de quelques principes :

- Le monde est constitué d'une grille de cellules en deux dimensions. (en théorie infinie, mais nous allons ici nous limiter à une grille finie)
- Le temps est divisé en étapes : on parle de générations. Le monde a un état à la génération N, puis à la génération $N + 1$, etc.
- À chaque génération, chaque cellule de la grille possède un état : soit elle est vivante, soit elle est morte.
- Le passage d'une génération N à une génération $N + 1$ est déterministe. C'est bien simple : si l'on vous donne l'état du monde à la génération N, en suivantl les règles, on ne peut arriver qu'à une seule

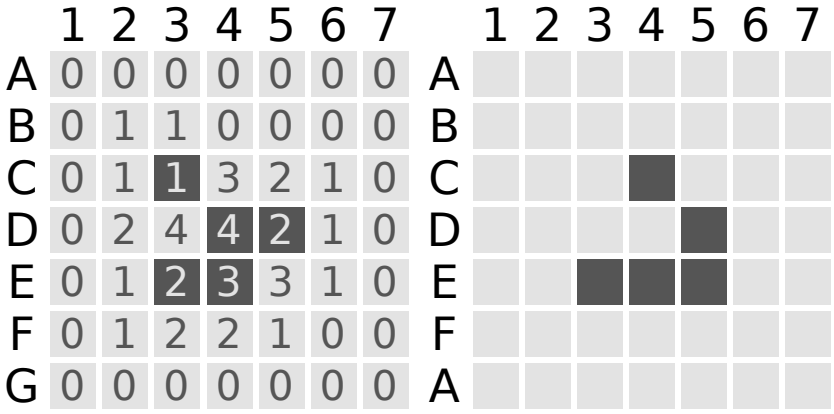
1. Game of Life, `http://en.wikipedia.org/wiki/Conway\%27s_Game_of_Life`

génération $N + 1$. Si on part toujours de la même génération, on arrive toujours au même résultat.

Ces règles sont elles aussi très simples :

1. Si une cellule est entourée de moins de deux cellules vivantes, elle sera morte à la prochaine génération.
2. Si une cellule est entourée d'exactement deux cellules vivantes, elle son état ne changera pas à la génération suivante.
3. Si une cellule est entourée d'exactement trois cellules vivantes, elle sera vivante à la prochaine génération.
4. Si une cellule est entournée de plus de trois cellules vivantes, elle sera morte à la prochaine génération.

Comme vous pouvez le constater, ces règles couvrent tous les cas, et elles ne se contredisent pas. Il ne manque plus qu'un petit exemple pour commencer :



Les cellules vivantes sont foncées, les cellules mortes sont claires. Les numéros sur les cellules indiquent (un peu comme au jeu du démineur) le nombre de cellules voisines vivantes.

Comparez les cellules vivantes avant et après :

– l'une d'entre elles (C3) meurt puisqu'elle n'est entourée que d'une seule cellule vivante (règle 1)
– une autre (D4) meurt puisqu'elle est entourée de plus de 3 cellules vivantes (règle 4)
– les autres ne changent pas d'état

Vous voyez, c'est très simple ! Le but de l'exercice est de créer un script Python afin d'étudier l'évolution d'un tel système.

2 Première étape : en console, gérer les structures de données...

2.1 Créer la grille

La première chose à faire dans le cas général, c'est de mettre au point la manière dont vous allez représenter vous données. On ne va pas aller par quatre chemins : le monde est ici une grille en 2D, nous allons donc le représenter par une matrice. Une matrice en Python ? Oui, nous allons faire comme dans pas mal d'autres langages : un tableau de tableaux (ou plutôt une liste de listes). Chaque élément de cette matrice sera un booléen représentant une cellule : True pour une cellule vivante, False pour une cellule morte.

Générer une liste de liste n'est pas bien compliqué. C'est la première chose à faire par le script : créer le monde à la génération 0 (génération initiale) :

```
grid = [
    [False, False, False, False, False, False],
    [False, True,  False, False, False, False],
    [False, False, True,  True,  False, False],
    [False, True,  True,  False, False, False],
    [False, False, False, False, False, False],
    [False, False, False, False, False, False]
]
width = len(grid[0])
height = len(grid)
```

```
2
```

2.2 Afficher la grille

Il est important de créer une fonction qui affiche la grille le plus tôt possible. En effet, la grille est l'élément central de script, par conséquent pouvoir afficher celle-ci à n'importe quel moment vous sera très utile au moment de développer vos algorithmes.

Afficher la grille n'est pas compliqué : il suffit de visiter toutes les cellules « dans l'ordre», et de les afficher en console. Indice : parcourir les éléments d'un tableau en Python, c'est itérer dessus (c'est-à-dire utiliser une boucle for)...ici on a des listes imbriquées.

2. Voir le fichier source game-of-life-grid.py

```
def print_grid(grid):
    # Insert some code here...
```

Si vous afficher un « O»majuscule pour une cellule vivante et un espace pour une cellule morte, vous obtiendrez quelque chose comme :

```
 O
  OOO
 OO
```

2.3 En avant marche !

La seule chose qui reste à faire est maintenant de créer une fonction `get_next_state` prenant en paramètre une grille à une génération N, et retournant la génération $N + 1$. C'est la partie poilue du script !

```
def get_next_state(grid):
    #Insert some code here...
    return next_grid
```

Le déroulement n'est pas trivial, mais décomposer des différentes étapes permet de passer cette difficulté. Globalement, l'algorithme demande d'itérer sur chaque cellule :

- Pour chaque cellule, on compte les cellules vivantes autour.
- En fonction de ce nombre et de l'état de la cellule, on détermine quel sera l'état de la prochaine cellule.
- On construit progressivement la grille à la génération suivante.

Itérer sur chaque cellule devrait vous être familier : il suffit de reprendre le code de la fonction `print_grid`.

Il faut aussi savoir compter les cellules voisines vivantes. On peut imaginer pour cela une fonction `count_alive_cells(grid, x, y)` qui compte les cellules vivantes autour de la cellule de coordonnées (x, y).

Il ne reste plus qu'à créer une fonction `get_next_cell_state(previous_state, alive_cells_count)`. Ainsi, à chaque cellule, il suffit d'exécuter quelque chose comme :

```
def get_next_state(grid):
    # Some code is missing...
    alive_cells_count = count_alive_cells(grid, x, y)
    new_grid[y][x] = get_next_cell_state(
```

```
            grid[y][x],
            alive_cells_count
        )
        # Some code is missing...
        return new_grid
```

3

3 Seconde étape : faire de bô dessins !

Deux possibilités s'offrent à vous, selon vos préférences :

- Soit vous préférez jouer plutôt « statique» : générer N images (`image-1.png`, `image-2.png`,...) correspondant aux N premières générations, puis afficher ces images avec le lecteur de votre choix. C'est le plus simple, nous présenterons ici la bibliothèque PIL [4].
- Soit vous préférez jouer dans le dynamique : afficher la grille dans une fenêtre directement depuis Python, puis avancer de génération en génération après la pression d'une touche du clavier, par exemple. Ça vous demandera un tout petit peu plus de code, mais c'est plus agréable à utiliser ! Sur ce coup, difficle de passer à côté de la bibliothèque pygame, *binding* Python de la fameuse bibliothèque en C : SDL [5] mais à vous de chercher comment faire !

Encore une fois, le déroulement du script est ici assez simple :

- On génère une image de la grille.
- On remplace la grille par la génération suivante.
- Et on recommence !

Générer l'image de la grille est un processus qui ressemble de très près au fonctionnement de la fonction `get_next_grid` : pour chaque cellule de coordonnées (x, y), on dessine sur l'image un carré entre les pixels de coordonnées (x, y) et $(x + c, y + c)$ (c étant la taille d'un carré), la couleur de chaque carré dépendant bien sûr de l'état de la cellule.

```
import Image
import ImageDraw

generations_count = 20
```

3. Voir le fichier source game-of-life-get_next_cell_state.py
4. Voir le site de PIL : `http://www.pythonware.com/library/pil/handbook/index.htm`
5. Voir la documentation sur le site de pygame : http ://pygame.org/docs/

```
cell_size = 10

grid = # ...
height = len(grid)
width = len(grid[0])

# Definition of functions (get_next_state, ...)

for i in range(generations_count):
    # Create a new image
    img = Image.new(
        'RGBA',
        (cell_size * width, cell_size * height),
        '#000000'
    )
    draw = ImageDraw.Draw(img)

    # ... For each cell in (x, y),
    # draw it on the image
    if grid[y][x]:
        color = '#ffffff'
    else :
        color = '#000000'
    draw.rectangle((
        x * cell_size, y * cell_size,
        (x + 1) * cell_size, (y + 1) * cell_size
    ))

    # ... then save the image and go to the next step
    img.save('grid-%d.png' % i)
    grid = get_next_grid(grid)
```

6

6. Voir le fichier source game-of-life-pil.py